Senior Project Proposal: ORCA

Logan Woodbury

Contact Information

Name	Logan Woodbury
Phone	(740)-837-8273
E-mail	Testare.i9z@gmail.com

Abstract

The goal of this project (Current codename: ORCA, or Orca) is to create a human-writable programming language. This means creating a language that can be written down using a normal writing utensil, then photographed and parsed by a program, and then interpretted as a script to execute, without needing a keyboard.

Background

OCR technology has been around for a while. From reading printed documents, numbers on cheques, and QR codes, technology has been around for a while that can interpret information. However, none of it has really been seriously used to aide development of software.

Likewise, many movies and depictions of the future involve humans communicating directly with machines using natural language. Many modern day products try to emulate this idea, such as Apple's Siri, Amazon's Alexa, and Window's Cortana; however these are all pretty limited in their communication. This becomes difficult as many things we take for granted in human conversations (Clarifying unfamiliar words, recognizing names, understanding context more than pragmatics, resolving ambiguity, etc) are difficult for machines to do, or resolve in a way humans find natural. This is good news for us software people, since our main job is to do that communicating with the machiens for people, but also kinda a bummer for this fantastic future we foresee.

I think that one way we might get around this in the future is to have a language created specifically for humans to communicate with machines, one that has grammatical constructs like a programming language to eliminate ambiguity in speech for the machine. Like esperanto, which has spread slowly despite its ideal of being a common tongue, learning a new language might not appeal directly to people immediately. But, as the feature gets incorporated into more and more machines to serve the "early adopters," the motivation to learn this language would increase dramatically, especially to learn the simple commands, as the world around us becomes more automated.

Now, this might be a bit of a pipe dream of mine, and one that I don't really intend to realize currently. However, I feel like maybe a step along this path is to create a distinct *written* language for computers, a written programming language.

I remember in high school, I did not have access to a laptop, but I would read out of my "Java for Dummies" book in class and would scratch out some basic code for me to try at home. I would often wish the words I had written would somehow compile and run on the paper, like some form of magic.

Now, I'll try my best to approximate that idea.

Description

Components

This project really has 3 components: The OCR (Reader), the graphemes (Characters), designing the language and making interpretters (Language)

Orca Reader

The purpose of this component is to create the software that can take a given image and...: * Determine what part of the image is the text * Fragment the text image into discrete parts (the graphemes) * Determine the ordering of the graphemes * Determine the identify of the graphemes * Put these together into a string

Hopefully this OCR can be flexible in performing these even under "extreme conditions."

There might be other requirements of the reader that unfold as the project continues. For instance, the writer might want to annotate his writing with words written in the native language (Comments). We might have to have a character that tells the interpretter to ignore all the writing past that point. Also, if the user is writing in pen, they might want to cross out characters that no longer matter.

Orca Characters

The Orca langauge will avoid using the typical Latin characters.

Since user programs are very persnickety about each individual character being correct, and many words are defined with unique character sequences (@my-ObjectVar2Iterator), it is particularly important that the characters chosen for this language are easily read by an OCR. The latin characters might be good at being recognized, but they were not designed for being read by machines. Also, there are many variations in the forms of these different characters (for instance, I myself don't write a the way it is presented on most computers).

Also, to distinguish text meant for machines from english text from humans, it makes sense to use different characters to represent the language. This helps separate ideas about English and ideas about this language, so that instead of imposing strange grammar and syntax on English (normal programming languages), we create text that helps tell the user that they aren't talking to something that thinks like they do. However, since it is impractical to generate an entirely new vocabulary for this language yet, we'll likely want there to be some sort of mapping from these new characters to English so that English vocabulary can be used to give meaningful names to functions, variables, etc.

However, one significant disadvantage of moving away from latin characters is that most people can write pretty quickly using these glyphs, since they always have to. We need to make sure that the characters are efficient to write, so that it doesn't cause users hand cramps before they write a single variable name. Ergonomics of the characters need to be considered. If we're particularly clever, considering the frequency of different characters and their likely sequences, so long as it does not significantly impact the accuracy of the OCR, we could create a more ergonomic language than written English. This consideration might be relegated to a stretch goal though.

Also, one more consideration: Printing. It might be a boon to have this language be printable so that a picture could be taken of a printed document and turned into a script there. In that case, The symbols might have to come from supported Unicode characters. It might be possible to just create a custom font for this language, however.

This last one also affects the decision of whether the read characters are stored using representative characters (Characters that resemble the ones written) or stored as ascii characters that map to the written characters.

Orca Language

While debately two different projects (designing the language/making interpreters), these two will likely be done simultaneously and iteratively, constantly changing the other, and so I consider them to be one component. When we are able to get the characters from the page to the computer, it will be necessary to determine the paradigm and mechanics of the language. Should it be imperative to serve a wider audience, and to more easily integrate with local computer services? Should it be functional to be easier to test explicitly? Object-oritented? Agent-oriented? Simple procedural to save on time? How do you create an interpretter for this type of language? Should it be compiled?

One major consideration for this project is that what the typical use case of this would be. I think the likely scenario is that this language will be used similar to a shell, using it to execute certain commands and define scripts that this user would want to run every now and then. This would be nice since you can write enough on a page to specify a small operation, and then use the computer only briefly to evaluate the photo and run it. However, as the project unfolds we'll have to research more into the use case for the language.

One ideal of this project is that there will be two interpretters for the language. This will help verify the language specification so that things that are ambiguous and assumed with one language aren't generally assumed.

Orca Environment?

Certain constraints exist in the physical world that don't exist in the computer world. One example of these is the length of paper: One sheet of paper has about ~33 lines. In comparison, even the smallest code files cover multiple hundreds of lines. Alternatives to compensate for these constraints might have to be created. One possibility would be to limit the content of a page: It must contain only complete function definitions or language directives. Then when we read photos in, they get added to an environment that holds all the functions and language directives scanned to that point. The functions can be tested from this environment, and if they work well they can be appended to other library files, or used as the main method of an executable. Then user defined functions could be added to their own standard "bin," and then invoked directly by writing the function name on a paper, taking a picture, and choosing to "execute" it.

This would certainly go along with the idea that the user is writing quick little "scripts" rather than huge software projects in this language.

Target Audience

The target user of Orca would be people who already have programming experience, or intend to learn. People who would be interested in running computer scripts while at their desk at school away from a computer.

Success Criteria

The overall quality of Project Orca can be determined by the following criteria: * Reader Accuracy - Is what is written accurately translated to the computer? * Reader Efficiency - How long does it take the Reader to read the characters? * Reader Flexibility - How many variables affect accuracy of the Reader? Can it read characters from an upside-down and slanted photo of crumpled paper with water stains, pencil smudges, and other doodles on it? On the other hand, can I cross out lines I don't care about anymore? * Character Flexibility - How many characters are available in the character set? * Character Ergonomics - How quickly can I write the characters on paper? Does it hurt my hand? Do I write the simplest character the most often? Does writing "flow" or is unnatural feeling? * Character Mapping - Are there counterparts to the standard ASCII characters? The standard Latin characters? Are there "false cognate" characters (characters that resemble latin characters but are not the counterpart)? * Language Appropriateness - Is the langauge a good match for the constraints of a written programming language, and those who might use one? * Language Functionality - What is the language capable of? Are there restrictions on what it can do? Is it practical to use? Is it extensible with libraries? * Langauge Explicitness - Is the language definition clear and unambiguous? * Holistic Functionality - Is Orca a tool others might use? * Holistic Motivation - Is Orca fun to use? Why would somebody use Orca? * Research Value - Was much learned through this project?

Scope

Goals

- Implement an OCR without using a standard OCR library for specifically created characters.
- Do some research and study into what would be a good character set according to specified criteria. Determine a character set using this criteria.
- Create a programming language using this character set, and an interpretter for this language.
- Write on paper a script for this language, have it read and turned into a working computer script.

Stretch Goals

- Maximize the success criteria
- Create a second interpretter for the language
- Create an environment to handle development and execution of projects in this language
- Library support

Non-goals

- Create a language to introduce people to programming
- Create the most user-friendly language

Significance

My reasons for believing this project to be significant are mainly summarized in the background section. I believe that creating explicit ways for humans to communicate arbitrarily with machines will be a key part in the future development of technology, since it already basically is one way that technology is progressing (Smart homes, voice commands on phone, etc). I believe this could be significant in that it pushes what has already been realized on this front, and could be a stepping stone for something that changes how society interacts with computers.

New Computer Science Concepts

I have done a little bit of OCR in the past, using primary component analysis to determine the digit in the standard MNIST dataset. This was my project in my Machine Learning class, and I used R. However, in that dataset all the characters were already oriented upwards and seperated into black-and-white images of a standard size. In this project, the input will be an unprocessed image, and I will have to separate the characters from the image myself and determine their order and orientation.

Not exactly the most "computer sciencey" concept, but I will also have to stretch my statastical and research ability as I determine which set of glyphs are best suited to the project: I will have to determine this based on the accuracy of the OCR in distinguishing these glyphs and the efficiency of writing the glyphs. Likewise, I'll probably have to compare different methods of character recognition for effectiveness with these different sets.

Finally, I will have to research different programming language paradigms and how to create a program language interpretter or two for that language. As BYU-I doesn't have much as far as a compiler class goes, this will likely be full of new ideas for myself.

Lastly, at least part of this project will be done in languages that are still a bit unfamiliar to me. The OCR will be implemented in Haskell, a language I've used a couple times but has a lot to offer in terms of unusual programming language characteristics.

Interestingness

If you can read the rest of this proposal and not get the impression of how interesting this is to me, then I don't think I can convince you in any lines here.

Tasks and Schedule

This project is to be completed over two semesters.

General tasks

- [**9-23 Oct, 4 hours**] Determine which are the main use cases for this software
- [**23 Oct-6 Nov, 4 hours**] Create vision for final product
- [**6-27 Nov, 8 hours**] Write Requirements Specifications

Total: 16 hours

ORCA Reader [First Semester]

- [**9-23 Oct, 16 hours**] Identification of what part of the image is text (Where text is in the image)
- [**23-30 Oct, 8 hours**] Normalization of text (Color normalization, orientation correction)
- [**30 Oct-6 Nov, 8 hours**] Separation of text into characters
- [**6-27 Nov, 16 hours**] Define basic algorithm for identification of characters, with acceptance of training data for the characters

Total: 48 hours

ORCA Characters [First Semester]

- [**13-20 Nov, 4 hours**] Prepare data for many different potential characters
- [**20-27 Nov, 4 hours**] Measure the effiency of writing the different characters
- [**27 Nov-4 Dec, 4 hours**] Run tests of accuracy on different characters sets for different sizes (It'll probably be about 4 hours to get automated tests set up, though the automated tests will likely take longer)
- [**4-11 Dec, 2 hours**] Determine the best character set (Highest size, accuracy, and effeciency). Probably with a few options, for the orca language

Total: 14 hours

ORCA Language [Second Semester]

- [**8 hours**] Research different language paradigms, determine some for the language
- [**16 hours**] Research how language interpretters work (Such as their data structures)
- [**4 hours**] Create the minimal language specification
- [**4 hours**] Create interpretter to determine if syntax is valid for the language (To catch "compiler error" type problems, can be run to test a script's validity without having to actually run the script)
- [**16 hours**] Create interpretter to execute an orca script file
- [**? hours**] Expand the language specification, while improving the interpretter(s)

Total: 48+ hours

Totals

126+ hours pending

(+ 34 already accomplished = 160+ hours)

Keep in mind these are all very rough estimates, as I have not done much work on a project like this before.

Required Resources with Costs

A camera is needed to get a picture of the handwritten text. (Already owned)

References

OCR

- Color Segmentation of Images Using K-Means Clustering With Different Color Spaces[https://www.cs.bgu.ac.il/~ben-shahar/Teaching/Computational-Vision/StudentProjects/ICBV121/ICBV-2012-1-OfirNijinsky-AvivPeled/report.pdf]
- Wikipedia: Maximally Stable External Regions [https://en.wikipedia.org/wiki/Maximally_stable_extrema
- Wikipedia: Feature Detection [https://en.wikipedia.org/wiki/Feature_detection_(computer_vision)]
- Wikipedia: Document layout analysis [https://en.wikipedia.org/wiki/Document_layout_analysis]
- Image Segmentation [https://cs.gmu.edu/~kosecka/cs682/lect-segmentation-part1.pdf]
- Font and background indepedent text binarization [https://www.researchgate.net/profile/Thotreingam_Ka and-background-color-independent-text-binarization.pdf]

Language

- Write you a Haskell [http://dev.stephendiehl.com/fun/index.html]
- Implementing a JIT compiled language using Haskell and LLVM [http://www.stephendiehl.com/llvm/]